Amazon International Apparel Sales Data Cleaning Challenge Report.

64basajjabaka@gmail.com • Basajja On X • Basajja On Github

December 24, 2024

Contents

1 Executive Summary

This report documents the comprehensive data cleaning process applied to the Amazon Apparel Sales dataset. The original dataset contained 37,432 entries with 10 columns, and through systematic cleaning and validation, the dataset was refined to 36,391 entries with 8 columns, ensuring data quality and consistency for downstream analysis.

2 Dataset Overview

2.1 Initial State

- Total Records : 37,432
- Total Columns : 10
- Data Types : 1 integer column (index), 9 object columns

2.2 Columns in Original Dataset

0. `index` - Integer index column
1. `DATE` - Date information (object type)
2. `Months` - Month information (object type)
3. `CUSTOMER` - Customer names (object type)
4. `Style` - Product style codes (object type)
5. `SKU` - Stock Keeping Unit codes (object type)
6. `Size` - Product sizes (object type)
7. `PCS` - Pieces/quantity (object type)
8. `RATE` - Price rate (object type)
9. `GROSS AMT` - Gross amount (object type)

3 Initial Data Preparation

At this stage, I standardised all column names to lowercase and replaced spaces with underscores (_). This was in order to have a consistent naming convention improve code readability and prevent errors from case sensitivity.

Next I dropped the `index` column as it was redundant hence remaining with 9 columns.

4 Data Cleaning By Column

4.1 Date Column

First I established that there was one row without an observation for the date feature and so I dropped it. Since it is the date column, my next step was to try to convert it to datetime type without coercing errors, but it throws a value error.

```
[7]
# convert the date column to datetime format
df1['date'] = pd.to_datetime(df1['date'], format='%m-%d-%y')
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[7], line 2
      1 # convert the date column to datetime format
----> 2 df1['date'] = pd.to_datetime(df1[          ], format=     %d     )

File ~\AppData\Roaming\Python\Python313\site-packages\pandas\core\tools\datetimes.py:106
   1066             result = arg.tz_localize("utc")
   1067 elif isinstance(arg, ABCSeries):
-> 1068     cache_array = _maybe_cache(arg, format, cache, convert_listlike)
   1069     if not cache_array.empty:
   1070         result = arg.map(cache_array)

File ~\AppData\Roaming\Python\Python313\site-packages\pandas\core\tools\datetimes.py:249
    247 unique_dates = unique(arg)
    248 if len(unique_dates) < len(arg):
--> 249     cache_dates = convert_listlike(unique_dates, format)
    250     # GH#45319
    251     try:

File ~\AppData\Roaming\Python\Python313\site-packages\pandas\core\tools\datetimes.py:435
    433 # `format` could be inferred, or user didn't ask for mixed-format parsing.
    434 if format is not None and format != "mixed":
--> 435     return _array_strptime_with_fallback(arg, name, utc, format, exact, errors)
```

Figure 1: Code Extract Showing the ValueError

On further inspection, I found out that it contained non-date values including: Customer names, Style codes, Months, and Column Names.

To solve this conundrum I embarked on the strategy below:

1. Identified misplaced data: Separated date entries into two categories:
    - Entries with numbers (actual dates or style codes)
    - Entries without numbers (customer names)
2. Style column imputation:
    - For date entries starting with a letter where style was null, filled style column with the date value
    - Result: 1,037 style observations were recovered
3. Customer column imputation:
    - For date entries without numbers where customer was null, filled customer column with the date value
    - Result: 2 customer observations were recovered
4. Special markers handling:
    - For entries containing 'SKU', 'Style', or 'CUSTOMER' in the date column, set customer to 'Unspecified'
    - Impact: This data recovery step significantly improved data completeness by utilizing misplaced information.

4.2 Months

On initial inspection of this column, I found 37,407 non-null entries with 24 missing values. I continued by examining rows with missing months values and found no meaningful data in other columns.

My action point here was to drop all 24 rows with missing months values resulting in 37,407 entries remaining.

Then i embarked on Format Standardization with 'Mon-YY' (e.g., 'Jun-21', 'Apr-22') as the Target Format. This is achieved through the following steps:

1. Format Detection and Imputation:
    - Created functions to identify correct format (`rformat`: 'Mon-YY') and date format (`wformat`: 'MM-DD-YY')
    - For rows with incorrect month formats, if customer column had correct 'Mon-YY' format, used customer value else used date value if date column had 'MM-DD-YY' format.
2. Date Format Conversion:
    - Converted date-like formats ('MM-DD-YY') to month format ('Mon-YY')
    - Created month mapping: '01'→'Jan', '02'→'Feb', etc.
    - Applied conversion: '03-09-22' → 'Mar-22'

```
# for the 24 null observations of months, inspect df1 to find out the
# corresponding observations in other features


df1[df1.months.isna()]
```

✓ 0.0s

	date	months	customer	style	sku	size	pcs	rate	gross_amt
18635	SKU	NaN	Unspecified	NaN	NaN	NaN	NaN	NaN	NaN
18636	JNE3826	NaN	NaN	JNE3826	NaN	NaN	NaN	NaN	NaN
18637	JNE3827	NaN	NaN	JNE3827	NaN	NaN	NaN	NaN	NaN
18638	JNE3828	NaN	NaN	JNE3828	NaN	NaN	NaN	NaN	NaN
18639	JNE3853	NaN	NaN	JNE3853	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...
18654	JNE3845	NaN	NaN	JNE3845	NaN	NaN	NaN	NaN	NaN
18655	JNE3846	NaN	NaN	JNE3846	NaN	NaN	NaN	NaN	NaN
18656	JNE3847	NaN	NaN	JNE3847	NaN	NaN	NaN	NaN	NaN
18657	JNE3843	NaN	NaN	JNE3843	NaN	NaN	NaN	NaN	NaN
18658	JNE3857	NaN	NaN	JNE3857	NaN	NaN	NaN	NaN	NaN

24 rows × 9 columns

Figure 2: Code Extract Showing the 24 Meaningless Observations

3. Format Cleanup:
    - Removed middle date portion from formats like 'Nov-03-21' → 'Nov-21'
    - Applied regex pattern matching to ensure 'Mon-YY' format.
4. Final Validation:
    - Correct format: 36,391 entries
    - Incorrect format: 1,016 entries

Through this, I dropped 1,016 rows that could not be transformed to the required format, leaving 36,391 entries with standardised 'Mon-YY' format in months column.

4.3 Customer Column

My initial inspection saw 36,391 Non-null entries and 158 Unique customers before cleaning.
For Date-like Value Removal, some month values ('Jun-21', 'Jul-21', etc.) were present in the customer column and replaced with "Unspecified" observation.
For whitespace Cleaning, stripped leading and trailing spaces and converted all names to lowercase. This resulted in 150 unique customers.

4.4 Style Column

Converted all style codes to lowercase. Stripped leading and trailing whitespace. This resulted into 36,391 non-null entries & 149 Unique styles after normalization.

4.5 SKU Column

This column had 1,434 missing values. I converted SKU to string type, lowercase and stripped whitespace. After conversion, NaN values became the string 'nan'.
I created a helper column `skuuuu` by extracting the last segment. For the 1,434 'nan' SKU observations I used the format: {`style`}-un-{`size`}.

4.6 Size Column

Replacements: '5XL' → 'xxxxxl', '6XL' → 'xxxxxxl', '4XL' → 'xxxxl'. I identified numeric values in size column and used the `skuuuu` helper column to infer correct sizes.

4.7 Numeric Columns

Columns `pcs`, `rate`, and `gross_amt` were converted to datatype `float64`.

5 Feature Engineering

5.1 Month Feature Creation

Extracted month abbreviation and year from `months` column to create `month` feature as a Period object. Dropped redundant 'date', 'months', and 'skuuu' columns.

6 Final Dataset State

6.1 Final Statistics

- Total Records: 36,391 observations.
- Total Columns: 8 columns.
- Data Types: 5 Object columns, 3 Float64 columns.

6.2 Final Column Structure

0. `month` - Period type (YYYY-MM format)
1. `customer` - String
2. `style` - String
3. `sku` - String
4. `size` - String
5. `pcs` - Float64
6. `rate` - Float64
7. `gross_amt` - Float64

7 Output Files Generated

- Intermediate Cleaned Dataset: `data/IntlsalesReport_Cleaned.csv`
- Final Cleaned Dataset: `data/IntlsalesReport_FinalCleaned.csv`

8 Key Challenges and Solutions

1. **Misplaced Data**: Problem: Customer names and style codes were in the date column. Solution: Relocated data using pattern identification.
2. **Inconsistent Formats**: Problem: Multiple date formats. Solution: Regex-based conversion.
3. **Missing SKU Values**: Problem: 1,434 missing values. Solution: Rule-based imputation.

9 Data Quality Metrics

Before Cleaning:

- Missing values: ∼1,040 rows (2.8%)
- Inconsistent formats: Multiple
- Data type issues: 3 numeric columns as strings
- Misplaced data: Present in date column

After Cleaning:

- Missing values: 0 (0%)
- Consistent formats: All standardized
- Data type issues: Resolved (all proper types)
- Misplaced data: Recovered and relocated

Data Loss:

- Rows dropped: 1,041 (2.8% of original dataset)
- 1 row: Missing date
- 24 rows: Missing months with no recoverable data
- 1,016 rows: Months in non-transformable format

10 Recommendations for Future Use

1. Data Validation: Implement checks at data entry.
2. Format Standards: Establish standards for dates and sizes.
3. Monitoring: Track data quality metrics over time.
4. Documentation: Maintain records of cleaning steps.

11 Conclusion

The data cleaning process successfully transformed a raw, inconsistent dataset into a clean, structured, and analysis-ready dataset. Through systematic cleaning, intelligent data recovery, and feature engineering, the dataset quality was significantly improved:

- 100% data completeness (no missing values)
- Consistent formatting across all columns
- Proper data types for all columns
- Recovered 1,039 misplaced data
- Feature engineering for time-based analysis

The cleaned dataset is now ready for exploratory data analysis, machine learning model development, and business intelligence applications.